

SCFE: Report 2 (Second Presentation)



Matthieu Stombellini
Mathieu Rivier
François Soulier
Rakhmatullo Rashidov



Contents

1	Introduction	3
1.1	Project subject reminder	3
2	Task progression	4
2.1	Core UI	4
2.2	Navigation modes	5
2.3	File I/O	7
2.4	User input	8
2.5	Tool & OS integration	9
2.6	Website	10
2.7	Documentation	11
3	Intentions for the final defense	13
3.1	Current state	13
3.2	Goal	13
3.3	Issues	14
4	Personal appreciation	15
4.1	Matthieu Stombellini	15
4.2	Mathieu Rivier	15
4.3	François Soulier	15
4.4	Rakhmatullo Rashidov	16
5	Conclusion	17
6	Appendix	18

1 Introduction

In this report, you will find details on the progress that has been made between the first defense and the second defense, with details on what was done and on the current state of the application.

The main idea for the second period was to transition from a solid and flexible basis to something more consistent: the actual application itself. This includes studying use cases as well as making interaction with the application as smooth as possible with decent cross-platform compatibility.

The second period also saw the creation and publication of the official website for Salamanders' Lab, which can be found at <https://salamanders.dev>, along with the documentation required to understand how to use the app.

Before reading this document, and especially the task progression part, do note there are two Mat(t)hieux involved in this project, Matthieu Stombellini (with two T's, who is also the project leader) and Mathieu Rivier (with only one T).

1.1 Project subject reminder

Salamanders' Console File Explorer, or SCFE for short, is a file explorer in the console that follows the following principles:

- Usefulness, providing helpful information and adding something to the table when compared to regular file explorers,
- Beginner-friendliness and intuitiveness, so that everyone has a chance of using the tool with all of its features with relatively low level of knowledge, especially when compared to regular CLI workflows,
- Power-user friendliness, so that more advanced users can get work done with SCFE,
- Approachability, by which we mean not in-your-face impressive but not letting you in without any clue of what you are supposed to do,
- Stability, as bugs are just an annoyance that no one appreciates, we will attempt to reduce them as much as possible.

It is built in C# using the .NET Core SDK, and, while its primarily targeted platform is Windows, it aims to be cross-platform. This decision was made both because it would increase the usefulness of the tool and because it is a necessity, as half of the members of our team use Mac computers.

The team behind this project is known as Salamanders' Lab and is made of four motivated students: Matthieu Stombellini (project leader), Mathieu Rivier, François Soulier, and Rakhmatullo Rashidov.

2 Task progression

The following part will present the progression of each task, also detailing who was in charge of which task.

2.1 Core UI

While not the main focus of this second period, this task mostly aimed at fixing bugs and stabilizing the base that was created. This base, named *Viu*, intends to be rather flexible with decent cross-platform compatibility.

2.1.1 Done

While building the main application, some unexpected bugs with the code we use to build graphical user interfaces in the console were encountered. The goal for this period was to, logically, fix them, as well as adding functionality here and there. Some components got “smarter” with a logic that properly handles being “squished”. The components would behave incorrectly if they did not have enough space around them: each component normally expects and requests a certain amount of space to be attributed. If the console was too small, it was impossible for the component to print itself properly, and a number of issues could happen: either the text would “overflow” to the next line, or the entire application would crash with a terrifying (but admittedly pretty stupid) out-of-bounds error. This was fixed and most components now support being “squished” and can show an ellipsis (...) at a configurable spot: strings can be cut from the left (...like this), from the right (like this...) or from the center (like...this), depending and what makes most sense. Another component that was improved was the table component (which is used as the file list in SCFE): it is now possible to scroll through the list if there is not enough space to display it.

The interface itself was entirely built using *Viu*: an excellent stress-test for the basis which grew stronger and more stable as a result. It was part of this task, but mostly consisted in putting everything together like Lego bricks.

The work on this task was entirely done by Matthieu.

2.1.2 To Do

From the second presentation to the final defense, the goal for this task is to once again fix bugs and ensure overall stability. Being the backbone of the entire app, this is crucial to ensure smooth user experience.

This task went from 70% to 90% during the second period. It is *on schedule*.

2.2 Navigation modes

This task was the most important one, as it consisted in actually implementing the various possible interactions that were planned.

2.2.1 Done

Two of the three planned modes were implemented: the NAVigation mode as well as the SEArch mode.

The NAV mode, which is more like usual graphical file explorers, consists in using the arrow keys and HJKL keys to move around the interface. Many shortcuts are already available to perform actions on the various files. The following are already implemented and fully functional:

- Pressing R to rename a file
- Pressing N to create a new file
- Pressing Shift+N to create a new folder
- Pressing the Delete key to delete a file
- Pressing C to copy a file, X to cut a file and V to paste a file

In order to make the application more fool-proof, destructive operations are either not performed (pasting where a file already exists with the same name) and show an error, or ask for confirmation. This way, deletion requires the user to type “yes” and press Enter in the text box to ensure that they do want to delete the file(s) they want.

The SEA mode also has these shortcuts, although they require using the Control key to access them (e.g. while you can simply press R to rename a file in NAV mode, you have to press Ctrl+R in SEA mode). This is because the regular letter keys do not redirect to the usually expected actions: they instead jump the focus to the text box at the bottom of the UI, and allow the user to enter letters for searching specific files. The user can then:

- Press Enter to either directly open the only file or folder that was found with this name
- If multiple files match the search keywords, pressing Enter will re-focus the file area instead and let the user choose which file they actually want.
- Pressing Escape cancels the search, empties the textbox and returns to the file list.

The **SEA** mode corresponds more to what console users expect: quick access to files by using the file names. It should be noted that the search performed is not recursive: only files in the current folder are searched. This is because the whole point of the **SEA** mode is to navigate through files as fast as possible by entering their name, and searching throughout the entire tree of files would be extremely tedious and slow down the application as a whole. In order to avoid this (and also avoid confusion from random files buried deep inside folders surfacing in a simple search), entering text while in **SEA** mode only searches in the current directory. In this way, it is better to think of the **SEA** mode as a “file view filter” rather than an actual heavy search mode.

Almost all of the actions described above are supported for multiple files: pressing Space or S allows the user to select files, and most actions will act accordingly (e.g. copying multiple files to the clipboard).

All of this was implemented using the Input map (turning keys into action names) and Action map (turning action names into actual handlers which perform the action) system we described at the previous defense. We now have a very flexible system which is able to hot-swap key bindings while leaving the actual actions untouched.

Moreover, actions such as pasting or deleting can take a lot of time. In order to properly support these, we are using multithreading to perform the operation on one thread while allowing the other threads to handle the user’s inputs freely. The thread in which the deletion happens then tells the other threads to either print a message, or reprint the entire folder if the content changed.

As a side note, an interesting thing to notice is that on the dev team we each have our own preferences on how to navigate in SCFE: some prefer the **SEA** mode for quick access to most files while others prefer the **NAV** mode for the ease of use and quick access to actions!

This task was done by both Matthieu and François. Since Matthieu was mostly responsible for the Core UI task and François for the File I/O task, Matthieu took care of linking the navigation mode to the interface components as well as the most complex things like multithreading, while François took care of linking the various actions to their implementation. All actions have two “branches”: showing the user that some action is happening or has happened, and actually doing the action on the files. Matthieu did the former, while François took care of the latter.

Examples of the application in action are available in the appendix.

2.2.2 To Do

The next remaining mode to implement is the **COM** mode, which will be a simple way to enter shell commands from inside SCFE. This mode is based on the text box component, and will be implemented in the same way as the various actions that require input from the users (e.g. renaming a file): type something in, then press Enter to confirm (or the Escape key to cancel). Because of this particular status, as it is not really a “navigation” mode per se, but instead an advanced action powerful action, it will not have a massive impact on the codebase from **SEA** and **NAV** modes, which provides a certain aspect of modularity to each mode.

This mode will be quite basic and will only be intended for simple commands. The output from the command will be displayed in the help message area. This mode is implemented across three areas: User Input, Navigation Modes, and OS & Tools Integration.

This task has gone from 20% to 70%, it is *on schedule*.

2.3 File I/O

The goal for this period in the I/O department was to expose more properties from files, as well as providing easy ways to sort and filter them.

2.3.1 Done

In order to have a more complete display for the various properties of each files, the current implementation had to be able to get more information from the files, e.g. access to the last modification date.

The system also needed to sort and filter files for a few reasons:

- The order of the files returned by the .NET APIs is not stable nor practical, and can be quite chaotic for large folders.
- *ALL* files are shown by default, meaning that system folders that are not even readable by anyone but the OS were visible, which ended up adding more clutter to the UI and causing more confusion as to which files were accessible and useful.

We, therefore, ended up with a pretty flexible system of re-routable filters and “smart sorting”.

- The base filter for showing files depends on a few variables, including whether hidden files should be shown or not, or whether we are in **SEA** mode and need to apply the search terms or not. Detecting hidden files is done in both the Windows and Linux/Mac OS way at the same time: the file can have the “Hidden” attribute (like on Windows) or can start with a dot (like on Linux and Mac OS).
- The sorting conditions we use take into account a few attributes: first whether the file is a folder or an actual file, and then the names of the files. This is not customizable at present, but we might allow the user to also choose whether they want to sort by size, modification date, or other.

Color schemes were also implemented. They simply consist in differentiating files (in white) from folders (in green) and hidden files (darker shades) from regular files (regular shades).

As a side feature, the File implementation we use now supports creating relative representations of links, which is explained in the “Tool & OS Integration” part.

This task was done by François.

2.3.2 To Do

Not much is left to do on this task. As usual, it will receive bug fixes if needed in order to maintain stability in the whole application. Our file wrapper around the .NET implementation might also need to expose more properties.

We might also provide a way to switch between a few sorting conditions, for example switching to sorting by last modification date or by size.

This has gone from 30% to 70% in the last period.

2.4 User input

The work for the User input task was minimal in this period, mostly consisting in including interactions where needed and fixing some bugs along the way.

2.4.1 Done

The text box at the bottom of SCFE is the basic way for the user to interact with the system. It is based on the same text box component as before, and most of the work done this time was correctly focusing and defocusing it in order to avoid having the user wrongly inputting text in it.

When the text box receives an “enter” key press or receives a new letter, it performs actions that depend on the state of the application: if an action is in progress (e.g. the user is entering the new name for a file because he triggered the “rename” sequence by pressing R), then the text box redirects the input to the current action handler, which is different for every kind of action. If no actions are in progress, then the text box either does nothing (in NAV mode) or filters the file view (in SEA mode) just like any search would. In NAV mode, the text box is usually not accessible, and if it is, it has no real action (except when actions like renaming are in progress of course).

Moreover, the “input system” (i.e. input and action maps) was hooked to the table in the application with this task, allowing the user to actually do things with his keyboard. However, actually defining the different bindings was not the goal of this task, as that was a job for the Navigation Modes task.

This task was done by Mathieu.

2.4.2 To Do

The User Input part has been going well and will face its final phase for the implementation of the COM mode. It will have to transmit information to an underlying shell, which might be tricky.

However, we think we either overestimated the amount of work required for the last part or underestimated the work that was previously done, as it would not really be a jump from 50% to 100%, but more from 75% to 100%. In any case, the task remains *on schedule*, but we are not entirely sure of whether our percentages accurately represent the work that was done or not.

2.5 Tool & OS integration

This task was moderately important, mostly consisting in providing basic ways to interact with the underlying OS and file system.

2.5.1 Done

In order to reduce the amount of clutter in the interface and avoid extremely long paths, in some cases, the path might be “relativized”. The only currently implemented instance of this is when the user is in a folder that is a subfolder of their home directory. For example: On Windows, if my user name is “MyName”, instead of “C:/Users/MyName/MyFolder”, SCFE displays “~/MyFolder” instead, which is more helpful (and uses a well-known standard of replacing the home folder of the user by ~). The folder’s name is deduced from

what the OS publicizes as the home directory, so it will work for any username under operating systems which are correctly supported by `.NET Core`.

Because a file explorer that cannot open files is quite useless, SCFE provides a lightweight integration with the underlying system through being able to open files. This uses the default action from the OS and opens the same app that would open if the user double clicks on the file in the system's file explorer.

This was done by Rakhmatullo.

2.5.2 To Do

The Tool & OS Integration part will be heavily involved in the last part of the project, as both light Git integration *and* the ability to input and run basic shell commands are planned.

This integration of outside tools will be done between the second and final presentation.

This task went from 0% to 30% in this period: it is *on schedule*.

2.6 Website

While some research had been done on what to do for the website, it is still an area that had not seen much progress. Because of this, it was one of the tasks with the most significant increase in progression.

2.6.1 Done

The website is currently live (and in HTTPS) at the following address:

<https://salamanders.dev/>

It has basic information on the project: what it is, who are the members of the team behind it, as well as a few placeholders where needed. It is fully responsive and works on mobile and ready to welcome more content. Its design is close to the one in our “graphical theme”: black and yellow accents, with the logos of both SCFE and Viu, the graphical library we created that is fully separated from SCFE's code base. We created the Viu logo specially for this website.

The website was done in raw (but clean) HTML and CSS using the Bootstrap toolkit and is hosted on OVH. The HTTPS certificate was obtained through OVH as well. This gives us total control over the website's content, although the creation and upload can be a little tedious at times.

The documentation is also hosted on the website and well integrated with the current theme. A Downloads page is also present but does not actually provide the downloads: these will be added before the last defense.

The work on the website was entirely done by Mathieu.

2.6.2 To Do

Now that the website basis is there, we need to actually populate it, both using the documentation (which we need to correctly integrate) and the various download links (downloading the app, the Viu demos, etc.).

This task went from 0% to 50%. It is *on schedule*.

2.7 Documentation

Documentation had slowly started for the first defense but has since seen more changes.

2.7.1 Done

Basic usage of the application has been documented and includes information on regular use of the app in expected workflows.

The documentation clearly explains the use of the **SEA** and **NAV** modes, ensuring to explain when and where to do something in a clear manner so as not to completely lose newcomers in useless details. It also provides a helpful load of key bindings, which are also available from within the application.

The application itself also has a healthy amount of information: the user can open special “option panels” which, instead of actually pressing keys, display all of the options available for a file, along with the key bindings that would allow one to perform the action. This help is dynamically created, so that if we want to have customizable key bindings in the future, the shortcuts shown in the application reflect the chosen key bindings – but as handy as it is, the help remains minimal, both because it should not interrupt the workflow and the console is a poor environment when it comes to reading text, mostly due to the limited amount of space and lack of proper font decorations.

The documentation that is outside of the application is written in Markdown and is available on the website under the Documentation tab. It is, of course, converted into HTML before being published, and some manual adjustments to the HTML code had to be made. Online documentation was done by

Rakhmatullo, while in-app documentation and help was done by Matthieu. The theming of the documentation on the website was done by Mathieu.

The documentation is available under the “Documentation” tab on the website.

2.7.2 To Do

The documentation will grow with the project and will be more consistent. The goal is to describe everything that can be done with the app and give hints on where to look if the user is ever lost.

This task went from 20% to 50% and is *on schedule*.

3 Intentions for the final defense

This part will go over the current state of the project, explain our goals for the final defense, and mention a couple of issues we are facing.

3.1 Current state

As a whole, the project has been going great. The basis was successfully and smoothly turned into a usable product, a full application with a solid yet flexible basis that gives us a new way to interact with our files: a mix between fully-featured graphical file explorers and lackluster but very fast console interactions.

Accessibility-wise, we have added all of the features thinking about the average Joe: what would a regular user think when he sees something? Unfortunately, as with anything that runs in the console, most reflexes will not come naturally, which is why we have an easy-to-read documentation in place to smoothly guide new users. We think that once they learn a few basic shortcuts (how to move around, how to switch modes, how to open some general options which display more advanced shortcuts...), they will be able to use everything else easily.

3.2 Goal

In order to meet the expectations we set in the book of specifications, we still have work to do in various ways:

- Try to improve cross-platform compatibility. SCFE has numerous issues when ran on non-Windows platforms and, while some have been fixed, others still need to be worked on.
- Thoroughly test and improve the usability aspect of the app. The app is made for productivity and always needs to be improved in that regard.
- Implement the remaining power-user features: mostly Git integration and the ability to launch shell commands.
- Overall add some polish to the app if and where needed.

Another goal for the last defense is to provide a full installable package which has all of the features we expect. The installable packages will come into multiple forms:

- With `.NET Core` bundled: we are expecting a heavy binary size, but this will be installable on any machine, with or without `.NET Core`

pre-installed. Moreover, we will be required to distribute one bundle per operating system, which is not very practical.

- Without `.NET Core` bundled: this will result into a smaller size for the package, but the application will rely on `.NET Core` being pre-installed on the user's machine, which is rarely the case since `.NET Core` is a relatively new framework.

The binaries will available be for Windows (`.exe`) and for all platforms (`.NET Core`'s `.dll` format). We might create packages specifically for Linux and Mac OS X, but these platforms will already be able to use the universal `.dll` format.

3.3 Issues

Unfortunately, we found out that some issues cannot be fixed until `.NET Core` 3.0 is released. These issues are almost entirely on non-Windows system, and will hurt cross-platform compatibility until the stable version of `.NET Core` 3.0 comes out. This is due to a bug in how UNIX consoles are handled: first, the characters that are entered are always displayed, even if you explicitly tell the console class you wish for the characters not to be printed, and the multithreading we use for parallel reading and writing to the console on UNIX systems simply cannot prevent the console from writing the characters to the output at this time. The next release from `.NET Core` fixes that, but in the mean time the non-Windows platform have glitchy interfaces.

The Pull Request related to our issue is available at the following link:

<https://github.com/dotnet/corefx/pull/35621>

It was recently merged into the recent project and is included into daily builds of the `.NET Core` framework, but these builds are by definition unstable and cannot be used for projects like ours until they hit a stable release.

4 Personal appreciation

This section contains the opinion of each member of the group on the project.

4.1 Matthieu Stombellini

Overall, I really enjoyed working in this period. My role as a project manager has taught me a lot about team work and how to handle this kind of project. It is still a new aspect for me which I am getting used to. I found the realization of the final product very satisfying, and I am eager to start working on more advanced features. In the mean time, some of the issues we have had, mostly with cross-platform compatibility, have been very annoying, and I have had periods of stress that were more intense than what would be desirable.

4.2 Mathieu Rivier

In essence, throughout this second period, we as a team have brought a lot of improvements to the SCFE application. We now have a working project that can already be used on a day-to-day basis. I think I am talking for the team when I say that we are all very proud of what we have achieved as of now and are thrilled to continue improving. On a more personal note though my work on the user input and especially on the website, have enabled our work to finally reach the light of day: it is a small part of the project but it remains meaningful to me. The team is great and I think that this project is teaching a lot to all of us. I think that this has been a great experience and taught me how to collaborate with others better and how to help others as well as accepting help when needed.

4.3 François Soulier

It is a great joy for me to see how much of a progress we were able to accomplish since the last deadline, and most importantly I am really proud of it. Working on the File IO task and improving it has been a very enriching experience - as it was for Navigation Mode. It allowed me to encounter some interesting programming issues, that were not easy to solve, and thus it brought me some precious experience. I feel very confident that the final result will be precisely what we expect it to be from the beginning: a tool that really simplifies one's life concerning file exploration. One point I would like to underline is the full cross-platform property of SCFE, that caught my eye in the first place, and it is a challenge we were able to face and overcome, so that makes not only me but all of us very proud of the entire project.

4.4 Rakhmatullo Rashidov

In general, the second period has been very productive as we have accomplished a lot, and we are all happy that everything is going according to our plan. I believe that working as a team not only improves our organizing and planning skills, but also, it helps to enhance our communications and problem-solving skills. Personally, I am very thrilled to do this project since I am learning new things every day. By working on Tool & OS integration and Documentation, I am getting more and more confident with C# and Markdown and I will do my best to contribute more to the project. Also, I really enjoy working in this team because whenever we have a problem, we solve it together as a team. All in all, I am sure that we will achieve what we planned to do by the end of the project.

5 Conclusion

Although some parts of this period were very frustrating and problematic, we have all managed to learn something from it while building a functional application we want to use. We are all very happy about the smooth transition we had, and are proud of the result.

While adding the final functionalities and finishing touches, we think our application will be ready to use with no particular issues for the last defense.

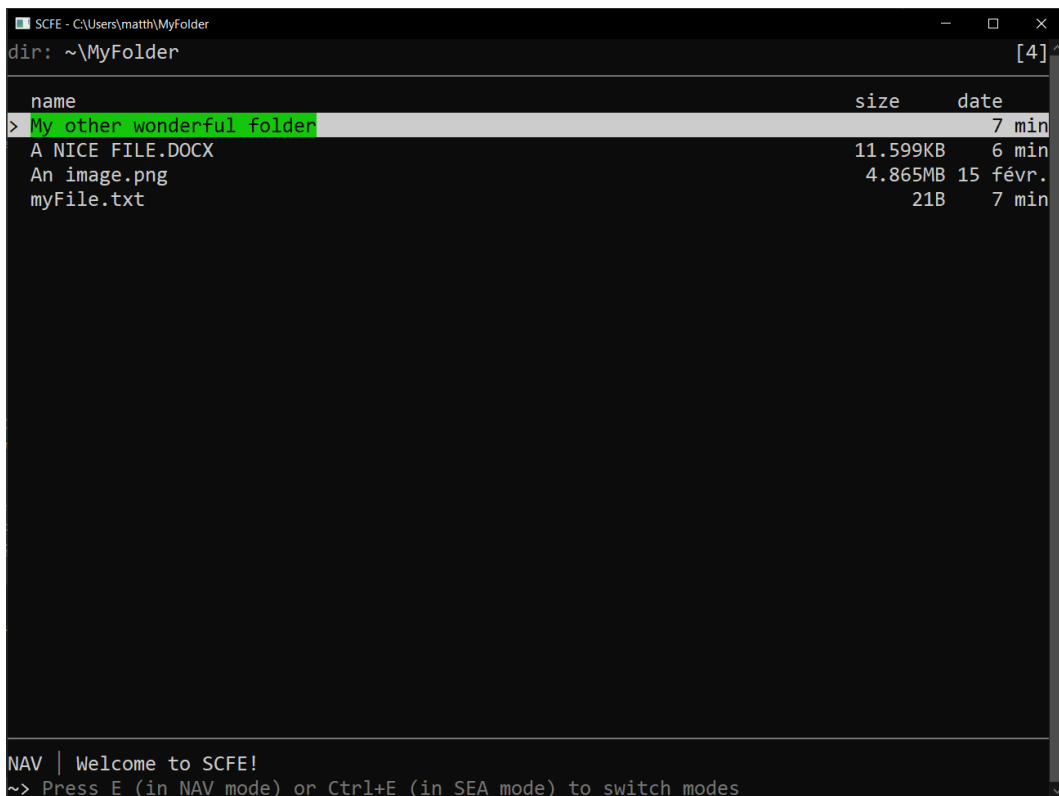
It is a very exciting turn for our project, where we have something usable. The next step is, of course, to make it even better and provide top-notch capabilities for hungry power users, which will be implemented through various quality-of-life improvements (Git integration and a way to launch shell commands for example).

We strongly believe that, because this application is already useful to some team members, there is a great chance that it will be useful for everyone.

6 Appendix

The following figures are screenshots from the application itself and the website.

Note that while all of the text we put in the app is in English, some strings (such as dates) are provided by `.NET Core`, and depend on the language of the system. Any French you see could also be in English, German or anything else depending on the language of the system. We decided against actively forcing them to English because we expect that the user of the system understands the language that he configured himself for his own system. Should that assumption be wrong, it would still be fairly easy to force the English language on all strings.



```
SCFE - C:\Users\mathh\MyFolder
dir: ~\MyFolder [4]
name                                     size    date
> My other wonderful folder             7 min
A NICE FILE.DOCX                       11.599KB 6 min
An image.png                           4.865MB 15 févr.
myFile.txt                              21B     7 min

NAV | Welcome to SCFE!
~> Press E (in NAV mode) or Ctrl+E (in SEA mode) to switch modes
```

Figure 1: The application on Windows

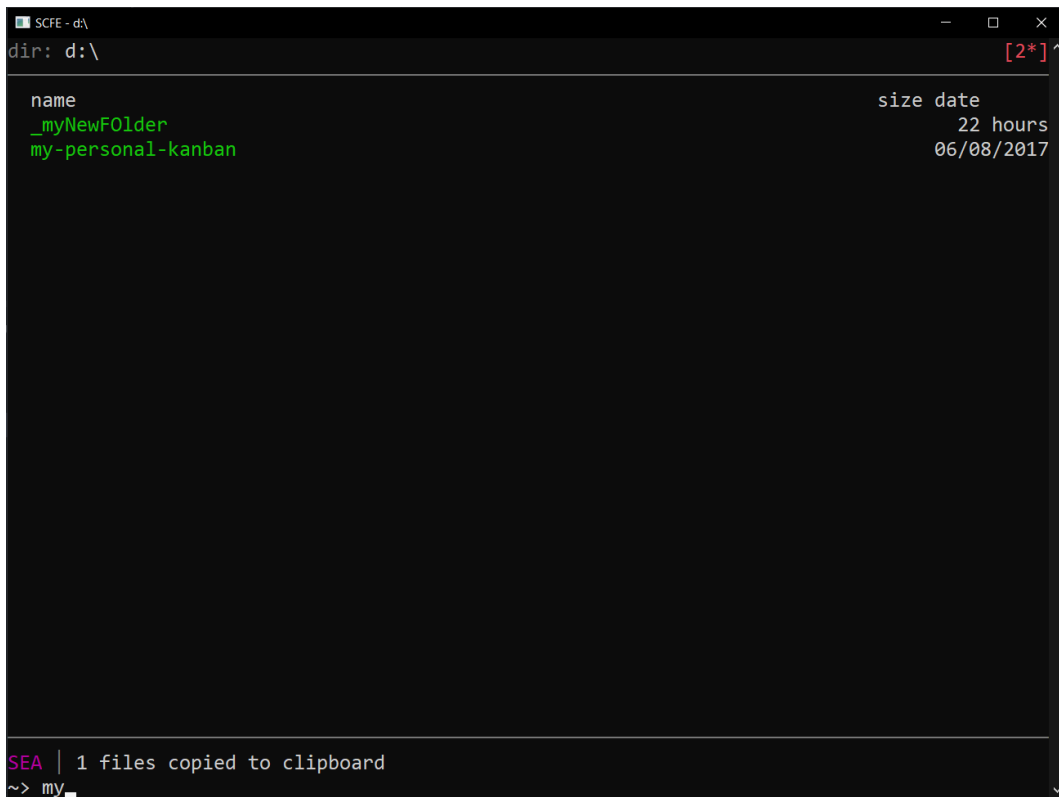


Figure 2: Using the SEA mode

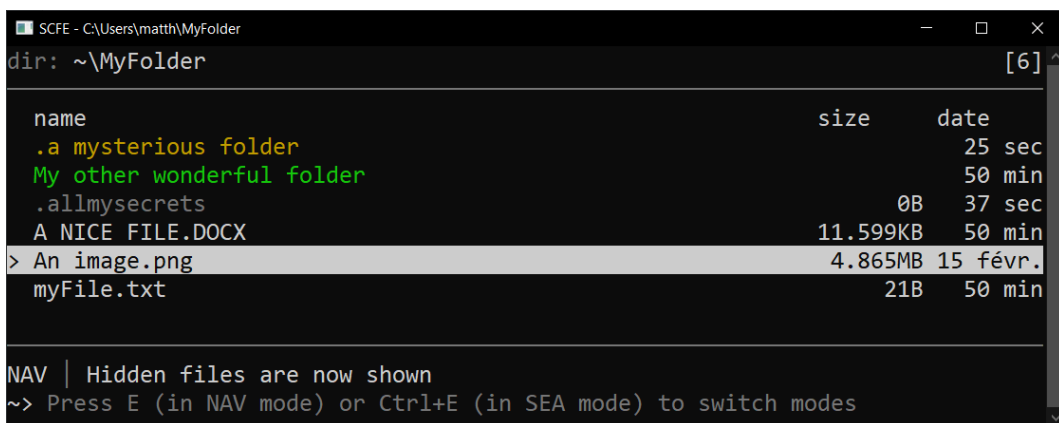


Figure 3: Displaying hidden files

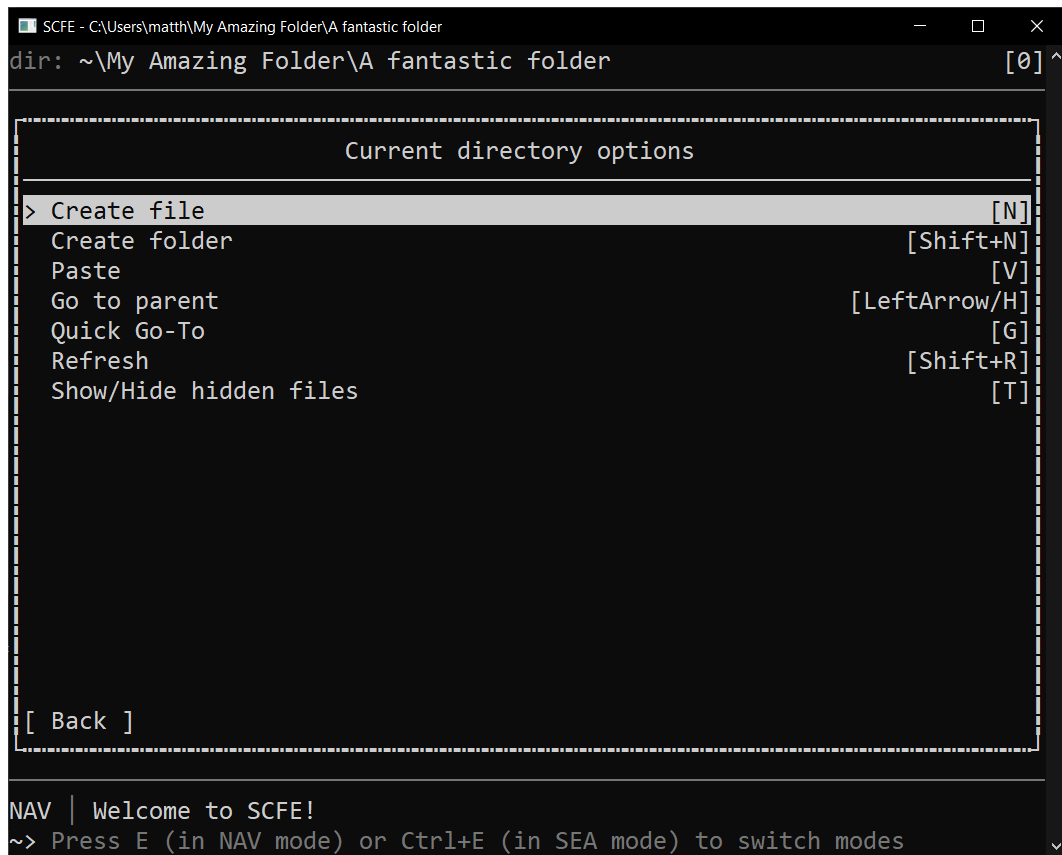


Figure 4: Directory options with the shortcuts displayed

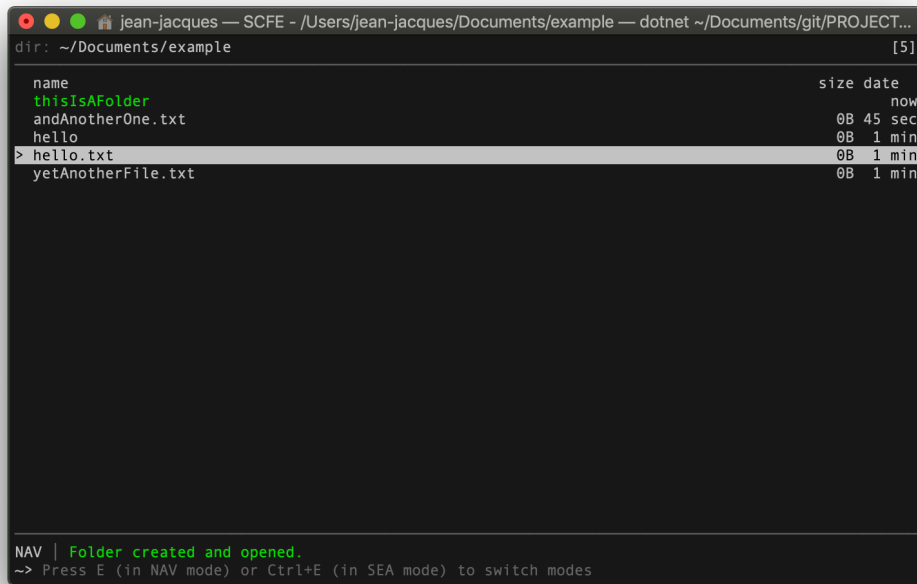


Figure 5: The application running on Mac OS X